John von Neumann Institute for Computing

Hybrid Parallelization for Interactive Exploration in Virtual Environments

Marc Wolter, Marc Schirski, Torsten Kuhlen

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr, F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 79-86, 2007.
Reprinted in: Advances in Parallel Computing, Volume **15**,
ISSN 0927-5452, ISBN 978-1-58603-796-3 (IOS Press), 2008.

http://www.fz-juelich.de/nic-series/volume38

Hybrid Parallelization for Interactive Exploration in Virtual Environments

Marc Wolter, Marc Schirski, and Torsten Kuhlen

Virtual Reality Group, RWTH Aachen University
E-mail: {wolter, schirski, kuhlen}@rz.rwth-aachen.de

Virtual Reality has shown to be a useful tool in the visualization of complex flow simulation data sets. Maintaining interactivity for the user exploring unstructured, time-varying data demands parallel computing power. We propose a decoupling of interactive and non-interactive tasks to avoid latencies. We introduce an optimized resampling algorithm for unstructured grids which remains scalable using hybrid parallelization. With a resampled region-of-interest, interactive execution of arbitrary visualization techniques is made possible. To validate findings, the user can access different error metrics made during resampling.

1 Introduction

With the growing size of scientific simulation output, efficient algorithms for the presentation of such results to a human user have become an important topic. The large amount of generated data must be explored by the user in order to extract the desired information. To compute visualizations on large, unsteady flow simulations with acceptable response times, parallelization of the corresponding algorithms has shown to be a worthwhile approach. Virtual Reality (VR) is a useful instrument for displaying and interacting with these visualizations. Time-varying 3D structures are perceived in a natural way, which provides a deeper insight into complex flow phenomena. However, the application of VR introduces an interactivity criterion to the system's response time. For direct interaction, Bryson[1] demands a response time of 100 ms to be considered interactive. Even with modern high performance computers, this threshold is not easily met for data sets of reasonable size. For image generation, dedicated visualization systems are used, which are normally locally and logically decoupled from high performance computing (HPC) systems. This introduces another source for latency, as all data generated on the HPC system has to be transmitted to the visualization system in order to be used for image synthesis.

To deal with this problem, we proposed a task distribution optimized for the user's interaction behaviour[2,3]. Tasks with frequent parameter changes are computed locally, while tasks for which parameters change less frequently or predictably are computed using hybrid parallelization on a remote HPC machine.

To render local visualization possible, we restrict it to a region-of-interest (ROI) reduced in size and complexity (as provided by a resampled Cartesian grid). Inside the ROI, the user can interactively explore the data using different visualization techniques. The resampling of a ROI for a time-varying dataset should be fast, but the interactivity criterion is relaxed somewhat, because this event is of significantly less frequent occurrence. Therefore, instead of parallelizing a specific visualization technique, we provide the user with arbitrary visualization tools for his data exploration by a general task distribution.

By distributing computational tasks onto processors and/or cores according to the data basis they operate on, our approach is especially targeted at the rising availability of multi-

core clusters. Distributed memory parallelization is used for tasks which do not share common data (i.e., data of different time steps), while shared memory parallelization is employed for tasks working on the same data (i.e., data of a single time step).

2 Related Work

Two main strategies for remote visualization are found in the literature: image-based and geometry-based approaches. Image-based remote visualization uses parallel computing to render the final images. This method is especially useful when handling very large data sets, as image space depends on viewport resolution only. Ma and Parker[4] describe software-based parallel rendering techniques for two visualization techniques, volume rendering and isosurface ray-tracing, for large unstructured meshes.

Strengert et al.[5] use hierarchical wavelet compression to keep the memory footprint low. They also integrate the user's viewing direction on a 2D viewport into the quality of the images rendered in parallel, providing some sort of implicit region of interest. Chen et al.[6] apply hybrid parallelization (MPI+OpenMP) on the Earthsimulator for volume rendering large data sets with additional vectorization of subtasks. They use dynamic load balancing between MPI nodes, as they distribute subvolumes among nodes.

However, the latency and restriction in viewport size generated by remote parallel rendering are not acceptable in an immersive virtual environment. A constantly high framerate (i.e., higher than 30 frames per second) in combination with head-tracked, user-centred stereoscopic projection is needed to maintain immersion. Due to stereoscopy and room-mounted displays, the amount of pixels to be covered exceeds normal desktop viewports by far.

In contrast to remote rendering, geometry-based remote visualization produces in parallel the requested results in form of geometry. Only the geometry is transmitted and is rendered locally at the visualization system. One of the first available systems for VR-based flow visualization was the Virtual Wind Tunnel and its follow-up Distributed Virtual Wind Tunnel[1]. The latter introduced a connection to a vectorized post-processing backend, which was then responsible for post-processing computations.

The Visualization ToolKit (VTK) provides different levels of parallelism[7]. Task, pipeline and data parallel execution of visualization pipelines are possible. They achieve scalable results while maintaining low complexity for the user. In addition, the VTK pipeline supports piecewise computation of results, which keeps the memory footprint low and enables the visualization even of large data sets.

3 Data Structures and Algorithmic Approach

As is typical for results of CFD computations, the problem domain is discretized in time and space, i.e. data is given on a number of unstructured grids. To allow fast interaction for the user, in our approach, data is to sent to the visualization system in the form of Cartesian grids of user-definable size and resolution. Thus, the unstructured source grids (or subportions thereof) have to be resampled into Cartesian grids. In previous work, we applied the algorithm provided by VTK for resampling[3]. This was the major bottleneck (i.e., $>$ 97 $\%$ was spent in the corresponding operations) even for rectilinear grids. As runtimes

increased significantly for unstructured grids, we now propose an optimized resampling algorithm for unstructured grids, which is easily parallelizable using OpenMP.

In a pre-processing step, the unstructured grids are converted into tetrahedral grids, which allow for a more efficient handling and cell search. Then, for every time step and for every target grid point, resampling is performed by locating the source grid cell which contains the target point, interpolating the data values of the points forming this cell, and storing this information at the corresponding point within the target grid. The most time-consuming part of this process is the cell search. For this, we employ a two-phase cell search strategy using a *k*d-tree and tetrahedral walks, providing significantly improved performance over the standard approach as implemented in VTK.

The basic cell search approach has originally been used for fast location of cells which contain seed points for interactively computed particle traces[8]. It comprises a broad phase using a *k*d-tree for the fast location of a grid point p_q located closely to the sought after target point q, followed by a tetrahedral walk from an incident cell to p_q to the final cell (see Fig. 1, left). However, as the tetrahedral walk can potentially fail if a boundary cell is encountered, we use an extended approach here, which considerably improves its reliability. The extension consists of restarting the tetrahedral walk from additional points p_{qi} and respective incident cells if it is not successful (see Fig. 1, right). These additional starting points are found by traversing the *k*d-tree upwards and using the encountered tree nodes as candidates. The query point q is considered as lying outside the source grid if and only if none of the tetrahedral walks is successful.

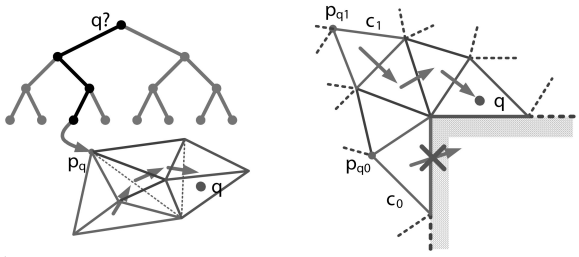


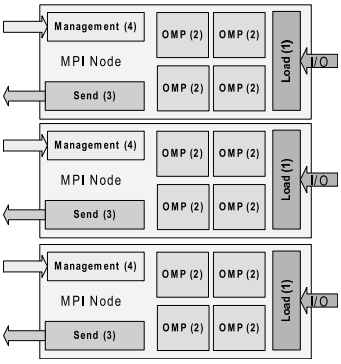
Figure 1. Point location is performed in a two-phase process consisting of a *k*d-tree search followed by a tetrahedral walk (left). Potential failures of the tetrahedral walk due to boundary cells are ameliorated by additional walks starting at supplementary candidate cells (right).

For an evaluation of the quality of the resampling process, a number of error metrics is employed. The corresponding information is then relayed to the user in order to let him assess the accuracy of the resampled grid, which directly influences the reliability of all further exploration efforts. For a global error estimation, the root mean square error is determined for all source grid points which lie within the boundaries of the target grid, followed by a normalization of these figures by a division through the data range of the involved points, thus leading to a relative root mean square error. In addition, the maximum relative error within this point set provides information about the upper bound of the interpolation error.

In order to determine the spatial distribution of local error, a relative root mean square

error is calculated for every target grid point as well, albeit with a restriction to localized vertex sets. It relies on those source grid points, for which the respective target grid point is the nearest neighbour in the whole target grid, and which lie within the boundaries of the target grid. This results in a spatial error distribution, which can be displayed to the user (e.g., via direct volume rendering – see Fig. 3, right) to allow for assessing the quality of the resampled grid. To confirm his findings, the user may request the ROI in its original structure, i.e. a subportion of the original data set, but hence potentially loses interactive exploration capabilities.

4 Parallel Data Extraction Phase




```
#pragma omp parallel shared(source, target) \
        shared(localmin[], localmax[])
{
  for (dimension k)
    for (dimension j)
      #pragma omp for nowait
      for (dimension i)
        p  = point(i,j,k)
        c  = locate_cell(source, p)
        if (c not found)
          c  = try_second_walk(source, p)
        if (c found)
          interpolate_point(c, p)
          update(localmin[], localmax[])
      endfor
    endfor
  endfor
}
update_globalvalues(localmin[], localmax[])
```

Figure 2. Left: Hybrid parallelization scheme. On each MPI node, at least four tasks are concurrently active: loading from filesystem (1), parallel computation using OpenMP (2), sending results to the visualization system (3) and managing control information (4). Right: Pseudocode for the resampling algorithm parallelized with OpenMP.

The parallel computation of the ROI is done using the Viracocha framework[9]. Single time steps of the data set are distributed using the scheduling scheme described by Wolter et al.[3]. Using this approach, the computational order of successive time steps is adapted to the user's exploration behaviour. Computed subtasks arrive just in time to be available for interactive computation. This is made possible by the constant animation duration of each time step and the predictability of successive time steps. However, a predictable and stable computation time per time step is crucial for a gap-free provision of a time-varying ROI.

Due to cell search and attribute interpolation being completely independent from each other for all sample points, a near-optimal speed-up can be achieved by distributing the corresponding computational load onto several processors or cores (see Section 6). However, since all computations for a single time step are performed on the same data basis, a shared memory system is the preferred computational environment, which makes OpenMP an ideal choice for the implementation (see Fig. 2, right). As an added benefit, our cell search approach results in a comparable amount of operations to be executed for every sample

point. This allows for very precise estimates of the time required for computation, which in turn helps prioritizing extraction operations for optimal user feedback.

A data manager component manages the amount of cached data on each node as well as prefetching required time steps.

For a fast ROI extraction, we apply a hybrid parallelization approach (see Fig. 2, left). Independent time steps are computed using MPI, which can be distributed across several nodes, as nearly no communication between nodes is necessary. A central scheduler process handles user requests and dynamic time step distribution.

To speed up the extraction for a single time step, pipelining and shared memory parallelization using OpenMP are applied on each MPI node (see Fig. 2, left). The three pipelining stages are (1) prefetching of the next time step data, (2) parallel extraction of the region of interest using OpenMP and (3) transmission (including serialization) of the simplified region. As time step data is predictable due to the continuous animation, simple OBL (one-block look-ahead) prefetching shows good hit rates for the prefetching thread (1). Preloading is overlapped with the current computation (2). The resulting Cartesian grid including error metrics must be serialized and sent over the network to the local visualization system (3). An additional thread (4) manages control messages (e.g., progress reports, update or cancellation of a task etc.). As all these threads need to communicate with other MPI nodes, a thread-safe MPI-2 implementation is required. Special care has to be taken not to waste CPU resources. Threads are suspended when their specific task is not required at the moment, but resume when a new task is available, which is realized with thread events. In addition, the MPI environment must be configured to prevent spinning of idle threads, which occupies CPU time otherwise available for the OpenMP computation. We currently use a spin time of one second, i.e. idle MPI calls check every second for new messages, which is enough for control messages.

The extracted region is transmitted to the local visualization system for further processing in the next phase.

5 Interactive Computation

Once the ROI is transmitted to the visualization host, the user can directly interact with the flow data and explore it using locally computed visualization algorithms. As an example, the user can specify particle seed points, seed isosurfaces, or position cut planes via an input device with six degrees-of-freedom inside the virtual environment. This information is directly used for a parametrization of the currently active visualization method, followed by an immediate computation and display of the result. The reduced grid data size (compared to the full-blown data set as stored on the HPC system) allows for visual feedback at interactive rates (i.e., < 100 ms)[3]. Any time the user wants to change the focus of the exploration, the ROI can be moved and/or resized, which results in a parallelized resampling of the data on the HPC system, again. If resampling times are low enough, the ROIs are computed just in time before they are displayed and move implicitly[3]. For larger resampling runtimes the user moves the ROI explicitly, which introduces a brief waiting time of a few seconds before he can continue with the exploration[2].

	16^3	32^3	64^3	80^3
# points	4096	32768	262144	512000
Vector error avg	6.69 %	3.47 %	1.48 %	1.08 %
Scalar error avg	3.62 %	1.69 %	0.68 %	0.51 %
Transmission time avg	0.097 s	0.107 s	0.733 s	1.432 s

Table 1. Global error values and ROI transmission times for three different resolutions of Cartesian ROIs.

	1 thread	2 threads	4 threads
1 node	10.335 s	9.174 s	8.75 s
2 nodes	8.437 s	7.887 s	7.69 s
4 nodes	7.663 s	7.326 s	7.287 s

Table 2. Total runtimes for resampling four time steps on the X2200 cluster, measured on the visualization system. The poor speedup is caused by transmission being the new bottleneck.

6 Results

We applied the system to explore a time-varying data set which consists of 100 time steps with approximately 3.3 million points per time step in an unstructured grid. The data show the turbulent mixing of a cooling jet in a hot boundary layer[10]. Measurements have been taken using a SunFire E2900 (12 UltraSparc IV dual core processors at 1.2 GHz, 48 GB of main memory), one SunFire V40z (four Opteron 875 dual-core processors at 2.2 GHz, 16 GB of main memory) and a cluster consisting of 16 SunFire X2200 nodes (two AMD Opteron 2218 dual core processors at 2.6 GHz, 16 GB of main memory, Gigabit Ethernet interconnect). The visualization system was connected to the remote HPC systems via a non-dedicated 100 Mbit/s network.

Table 1 shows the relative mean square error of a ROI with several resampling resolutions. A very turbulent region was chosen as region of interest in order to give an idea about worst case errors. The average resampling error is quite low and drops significantly with growing resampling resolution, as expected. Figure 3 (right) shows an example of a direct volume visualization of the local error. The lower part has a much higher local resampling error, as this part has a higher cell resolution in the unstructured grid. While the global error gives a general idea about the resampling error, the researcher should check for the local error to validate results. On the downside, the transmission times for the Cartesian grid grows linearly with the number of resampled points. The computed Cartesian grids contained one vector and one scalar field, both with double precision. The high transmission times are explained with the non-dedicated, slow network and the usage of reliable TCP without any compression.

Figure 3 (left) shows the speedup of the resampling algorithm using OpenMP. The parallelization scales well, as single point searches are independent from another and all data resides in shared memory. As an additional benefit, the root mean square error of the resampling duration decreases, which makes it easier to estimate runtimes. For the speedup measurement, the SunFire V40z with eight cores was used, which explains the performance drop at eight OpenMP threads, as one CPU core was reserved for the scheduling

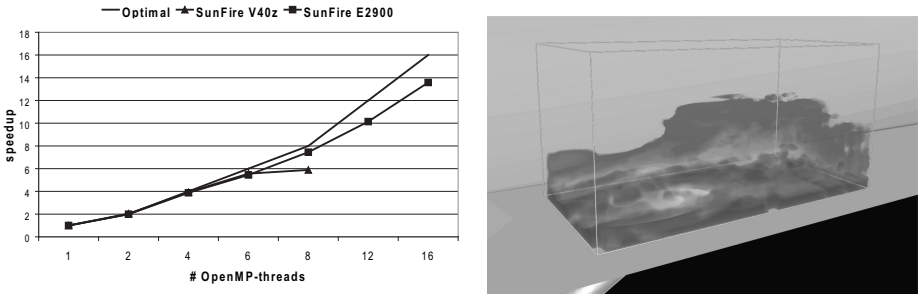


Figure 3. Left: Speedup for OpenMP parallelization of the resampling algorithm. Right: Displaying the spatial error distribution via direct volume rendering allows the user to assess the quality of the resampled grid ($64\times32\times32$). Local error values between 0% (transparent) - 10% (dark) are depicted.

component (cf. section 4).

Table 2 depicts the total runtimes for hybrid parallelization of four time steps. A region was resampled on a 80^3 resolution using up to four MPI nodes and up to four OpenMP threads. The runtimes were measured on the visualization system, and therefore depict total user waiting times. While a performance gain is achieved due to the speedup of the resampling algorithm itself, the total runtime savings do not scale just as well. Further analysis has shown that with the highly reduced resampling time two new bottlenecks arise: file I/O and data transmission to the visualization system. In the shown measurements we used already prefetched data in the data manager's cache, as the difference in loading and computing a time step was too long for efficient prefetching. For more efficient file I/O, external memory algorithms will be implemented as future work. MPI's parallel I/O is not useful in our case, as single files are processed by single MPI nodes. For the transmission times, which exceed computation time for all resolutions, compression in combination with a faster and dedicated network could improve the results. We have already measured high throughput (660 Mbit/s on a dedicated gigabit Ethernet) using such a network in previous work[11].

7 Conclusion

Hybrid parallelization minimizes user waiting times for newly extracted ROIs within large data sets considerably, thus speeding up the exploration process. By constraining the response requirements to the user's actions inside the ROI only, our approach allows for an interactive exploration even of large data sets in virtual environments. A drawback of this method in general is the user's restriction to a spatially bound region of interest. Although this corresponds to the common exploration pattern, the user does not get any information outside the specified region. The shift from directly parallelizing the visualization method to considering the region extraction as the target of parallelization leads to an efficient and scalable parallel algorithm. However, for large data sets and highly resampled regions, file system and network transmission emerge as new bottlenecks.

Acknowledgements

The authors would like to thank the Department for Mathematics CCES for providing access to the Opteron cluster.

References

1. S. Bryson and M. J. Gerald-Yamasaki, *The Distributed Virtual Windtunnel*, in: Proc. IEEE Supercomputing '92, pp. 275–284, (1992).
2. M. Schirski, C. Bischof and T. Kuhlen, *Interactive exploration of large data in hybrid visualization environments*, in: Proc. 13th Eurographics Symposium on Virtual Environments and 10th Immersive Projection Technology Workshop, pp. 69–76, (2007).
3. M. Wolter, C. Bischof, and T. Kuhlen, *Dynamic regions of interest for interactive flow exploration*, in: Proc. Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '07), pp. 53–60, (2007).
4. K.-L. Ma and S. Parker, *Massively parallel software rendering for visualizing large-scale data sets*, IEEE Computer Graphics and Applications, **21**, 72–83, (2001).
5. M. Strengert, M. Magallon, D. Weiskopf, S. Guthe and T. Ertl, *Hierarchical visualization and compression of large volume datasets using GPU clusters*, in: Proc. Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '04), pp. 41–48, (2004).
6. L. Chen, I. Fujishiro and K. Nakajima, *Optimizing parallel performance of unstructured volume rendering for the earth simulator*, Parallel Computing, **29**, 355–371, (2003).
7. J. Ahrens, C. Law, W. Schroeder, K. Martin and M. Papka, *A parallel approach for efficiently visualizing extremely large, time-varying datasets*, Tech. Rep. LAUR-001630, Los Alamos National Laboratory, (2000).
8. M. Schirski, C. Bischof and T. Kuhlen, *Interactive particle tracing on tetrahedral grids using the GPU*, in: Proc. Vision, Modeling, and Visualization (VMV) 2006, pp. 153–160, (2006).
9. A. Gerndt, B. Hentschel, M. Wolter, T. Kuhlen and C. Bischof, *Viracocha: An efficient parallelization framework for large-scale CFD post-processing in virtual environments*, in: Proc. IEEE Supercomputing '04, (2004).
10. P. Renze, W. Schroeder and M. Meinke, *LES of film cooling efficiency for different hole shapes*, in: 5th International Symposium on Turbulence and Shear Flow Phenomena, (2007).
11. T. Duessel, H. Zilken, W. Frings, T. Eickermann, A. Gerndt, M. Wolter, and T. Kuhlen, *Distributed Collaborative Data Analysis with Heterogeneous Visualisation Systems*, in: Proc. Eurographics Symposium on Parallel Graphics and Visualization (EGPGV '07), pp. 21–28, (2007).